2010 3rd International Conference on Computer and Electrical Engineering (ICCEE 2010)
IPCSIT vol. 53 (2012) © (2012) IACSIT Press, Singapore
DOI: 10.7763/IPCSIT.2012.V53.No.2.32

Design and Implementation of a Parallel Thiessen Polygon Generator

Yongyuan Shen[+] and Buyang Cao

School of Software Engineering, Tongji University

Shanghai, China

Abstract—To speed up the generation process of Thiessen Polygons from mass 2-D points, a parallel generator is designed and implemented. The generator efficiently parallelizes the Thiesen polygon generation procedure and achieves the same results as the non-parallel version does. Experimental results indicate the generator is able to speed up the generation procedure significantly. The generator could be applied to large-scale problems or problems with real time requirements.

Keywords-component; Thiessen Polygon; Delaunay Triangle; Parallel Computing; Quad-Edge

1. Introduction

In many Geographic Information Systems, the Thiessen Polygon (Also can be called as Voronoi Diagram) is applied various spatial analysis procedures. To deal with large number of input 2D-points, the traditional sequential generator's speed is very slow. But meanwhile, the multi-core computer is more and more common in our daily life. So a natural idea is that a parallel generator is needed for a better computation performance.

Building Delaunay Triangulations is the base to build Thiessen Polygons. They can be converted to each other according to a certain conversion rule. There are several ways to generate the Delaunay Triangulation. As it [1] points out, in sequential environment and for both uniform distribution input points and non-uniform ones, the Dwyer algorithm [2] is better than others in terms of performance when the input points number is under 2^{16}. This algorithm has a handy partition characteristic for parallel processing. Also the partition to sub points set is benefit to control the input points into its best suitable problem scale. This paper designs and implements the parallel algorithm based on it to build a parallel Thiessen Polygon generator.

2. Algorithm Flow

The common algorithm flow of a Thiessen Polygon generator can be described as followed.

- Read the input data set. Exclude extra points with same coordinates in this points set. Then calculate the Minimal Bounding Rectangle (MBR) of the points.
- Add four extra points far away from the corners of the MBR. The four points constitute a new rectangle which is 21 times length and width of the MBR.
- Delaunay triangulation.
- Convert the Delaunay Triangles to Thiessen Polygons.
- Clip the Thiessen Polygons in to a suitable rectangle area, output the final results.

Because the input points' file format is ESRI Shape File that takes Quad Tree as its own internal data structure, so the MBR can be retrieved directly. While excluding the duplicate pointes, if the generator takes the linear structure as the search structure, it has to repeat searching the whole array or list for many times. When the number of input points is very large, the performance is very poor. Therefore the generator sets a

[+] Corresponding author.
E-mail address: shenyongyuan@gmail.com

hash table that takes the X coordinates as keys and Y coordinates' list as value sets. The experiments show that when the input number is 20000, this improvement can achieve about 30 times speed-up. For the clip procedure of Thiessen Polygons, the generator deploys the Sutherland Hodgman algorithm. Because the clip process for each polygon cell is independent, parallelization is very easy to apply.

Empirically, for a large input data set with more than 10000 points, steps of Delaunay triangulation and Thiessen Polygons conversion occupies about 90% of total calculation time. Hence parallelizing the two steps is the key points to improve the generator's performance.

3. Tessellation

For converting rapidly from Delaunay Triangles to Thiessen Polygons, the generator uses the Quad-Edge [3] as its foundational data structures, in which every Quad-Edge cell stores two main edges (e0&e2) and two dual edges (e1&e3) and all the edges are stored in sequence according to clockwise. In this case, the generator can access these edges directly without extra sorting operation. For each vertex, the dual edges around it are the edges of the corresponding Thiessen polygon cell. Because the output file is also an ESRI Shape file, it requires the polygon's edges be sorted clockwise before polygon data it is stored. Therefore it is the same as Quad-Edge's. In this case, the conversion between Delaunay Triangle and Thiessen Polygon is very fast and convenient. And more importantly, the conversion for each cell is independent and therefore this procedure can be parallelized as well.

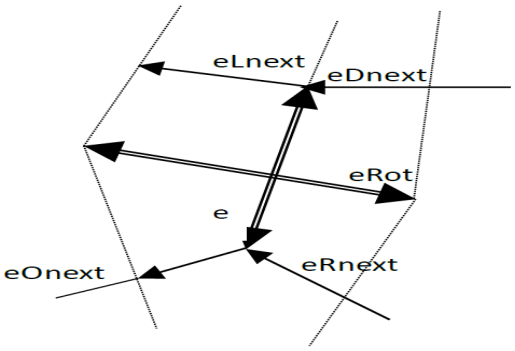

Fig.1. Quad-Edge Data Structure

The structure of Quad-Edge is very simple, it can describe the topology of spatial data model and geometric properties only using points and directed edges. In the Quad-Edge structure, Org stands for the original point of the current directed edge. eRot stands for the edge that rotates current edge 90 degree counterclockwise. eOnext means the next edge according to counterclockwise sequence based on current edge. In addition, it also defines the concepts of End Point, Left Part and Right Part. The structure specifies some basic operations like Make, Connect and Splice etc. The detail can be found in [3].

The experiment indicates that the performance is improved greatly after Quad-Edge data structure instead of traditional vertexes having been applied. Compared with the total execution time, the step of conversion can be almost ignored. So a more efficient parallelized Delaunay Triangulation step is an important step for the generator's performance improvement.

4. Parallel Delaunay Triangulation

The algorithm of this paper first divides the input points into several subareas according to the MBR of input points. The number of subareas to be created is followed the advice of [2]. Then the generator solves each subarea separately and combines these areas together. Because the partition and solution of each area is individual, the parallelization is feasible for this process. And it is worth to be pointed out, the input points' data set may not be uniformly distributed, it requires a method to balance the work load of each working thread. An easier way to solve this problem is to generate many threads whose number is far more than the calculation units such as the number of CPU cores. With the help of a thread management pool, the generator can keep all the calculation units working most of the time so that all CPU cores will be fully used.

The generator employs the divide-and-conquer method for each subarea. For an area with only two or three points, the generation is very simple. The program can simply links these points with directed edge to build up Delaunay Triangle. If the points number of this area is large than three, it is required to perform a partition operation and calculate each area separately, and finally to combine all subareas.

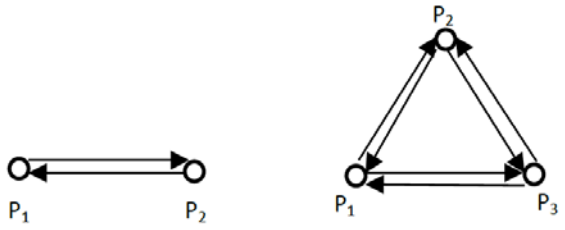

Fig.2. Delaunay Triangle generation for two or three points

The generator uses the following steps to combine two subareas. Firstly, it gets the bottom sideline and top sideline of two sub convex hulls, these two sidelines can be marked as B_1B_2 and T_1T_2.Then it finds a point B_3 beyond B_1B_2. If B3 belongs to left sub convex hulls, the new bottom sideline is B_2B_3, but if B_3 belongs to right sub convex hulls, the new bottom sideline will B_1B_3.To find a better B_3 point, it's advised to choose right border points of left sub convex hulls or left border points of right sub convex hulls, and the point should be lowest in the Y coordinate. A better B_3 point can reduce the optimization steps cost shortly after. Also the generator should ensure that the new bottom sideline should not intersect with old one. The step repeats till the bottom sideline is coinciding with top bottom sideline. But now the result is not the Delaunay Triangle, a Local Optimization Procedure should be applied to the intermediate result. The basic idea of this procedure is to combine the triangles with shared edges to a quadrangle. And then it exams the fourth point lies inside the triangle's circumscribed circle which made up by the other three points or not according to the Largest Empty Circle Criterion. If the result is true, then switch the diagonal of the quadrangle to finish the optimization procedure.

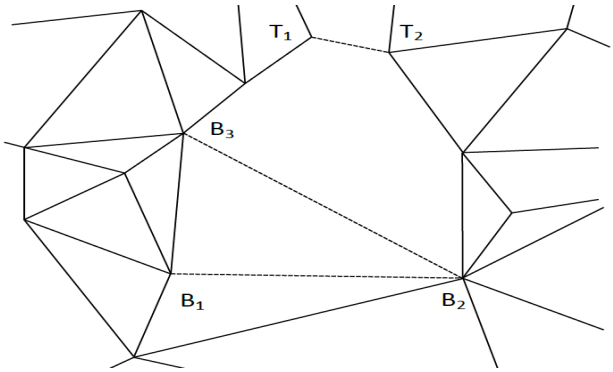

Fig.3. Cobination Procedure

If there are many points very close to the same vertical line, the algorithm may not be able to ensure its robustness because in this situation the sideline may be vertical to the baseline direction. To avoid this problem, the generator can switch the division direction while in every looping iteration. It means doing one division along the X axis and doing the next division along the Y axis till the end. This method can avoid processing area that with too high aspect ratio of the length and the width.

Furthermore, if there are three pre-defined points, to judge the fourth point is located inside or outside the circle made up with the three points, a common method is to solve the intersect point of perpendicular bisectors of every two points' connection line. This point is the center point of the circumscribed circle. Then it will compare the length of this center point to the fourth point with the radius of the circumscribed circle. But the generator should be aware that if the pre-defined points are at the same line, the situation should be handled separately.

It can be proven that the time complexity of searching the bottom and top common tangents belonging to two convex hulls is O(n). So is the step for the edges connection and optimization. The time complexity of the divide step can be regarded as O(1). If the total time complexity of the whole algorithm is set to T(n), it can be inferred that T(n)=O(nlogn).

5. Computinal Resultes

Based upon the previous discussion, a parallel Thiessen Polygon generator is designed and implemented. The generator is developed by .Net Framework 4.0(C#) platform. In our benchmark, ESRI ArcGIS Desktop 9.3 is also employed here, which is one of most famous and leading GIS application software. And it is known that the internal algorithm to generate Thiessen Polygon is sequential. The Thiessen polygon for every input test data is created by ArcGIS as well. One reason for this is to verify the result correction; the other is to compare the processing speed between the generator and ArcGIS.

The test environment of this paper is as followed, CPU: Intel Core 2 Quad Q9400 (2.66Ghz*4, 1333Mhz FSB,6M L2 Cache);Memory: 2G(DDR3 1333Mhz); Operating System: Windows 7 (32bit). Each group's input data is distributed in uniform distribution and produced by random function. And all the tests would be executed for five times and take the average execution time as the final result. The detail benchmark result is shown in Table.1 and the time unit is second.

Here is also a speed-up ratio comparison in Table.2 which compares with the parallel version and sequential version benchmark result.

TABLE I. EXECUTION TIME

Points Number	**Generator**			
	ArcGIS(A)	***Single Thread(S)***	***Dual Thread(D)***	***Quad Thread(Q)***
10000	16.261	1.549	0.738	0.526
20000	27.973	5.692	2.096	1.204
40000	51.521	22.681	7.095	3.132
80000	98.942	94.507	26.262	9.570
160000	198.211	393.890	102.933	30.583

TABLE II. SPEED UP RATIO

Speed-up Ratio	**Points Number**				
	10000	***20000***	***40000***	***80000***	***160000***
Q/S	2.944	4.725	7.241	9.875	12.879
D/S	2.099	2.715	3.197	3.599	3.827
Q/A	30.914	23.233	16.449	10.338	6.481

6. Conclusion

From the experimental result, it can be recognized that the parallel Thiessen Polygon generator has the superiority to the sequential version significantly. Also compared with the commerce software ESRI ArcGIS, the parallel Thiessen Polygon generator has a very clear advantage in calculation speed in parallel environment. So the parallel Thiessen Polygon generator achieves satisfactory speed-up ratio and is full of piratical value.

7. Acknowledgement

We thank for Qimeng Fan, Yuanchao Shi, Huihui Huang, Xingang Fan and Jinghui Li’s help for this paper.

This paper is sponsored by Key Project of the National Eleventh-Five Year Research Program of China (2006BAJ11B).

8. References

[1] P. Su and R. L. S. Drysdale, "A Comparison of Sequential Delaunay Triangulation Algorithms," Computational Geometry: Theory and Applications. Amsterdam, vol. 7, issue 5-6, pp.361-385, April 1997.

[2] R. A. Dwyer. "A Faster Divide-and-Conquer Algorithm for Constructing Delaunay Triangulations," Algorithmica. New York, vol 2, numbers 1-4, pp. 137-151,November 1987.

[3] L. Guibas,J. Stolfi. "Primitives for the manipulation of general subdivisions and the computation of Voronoi," ACM Transactions on Graphics (TOG).New York, vol 4, issue 2,pp. 74-123, April 1985.

[4] G. Blelloch,G. Miller,J. Hardwick and D. Talmor."Design and Implementation of a Practical Parallel Delaunay Algorithm," ALGORITHMICA Special Issue on Coarse Grained Parallel Algorithms. New York, vol 24,pp. 243–269, August 1999.

[5] Nikos Chrisochoides and Demian Nave."Parallel Delaunay mesh generation kernel," International Journal for Numerical Methods In Engineering.New York, vol. 58,pp. 161-167,July 2003.

[6] Martin Isenburg,Yuanxin Liu,Jonathan Shewchuk and Jack Snoeyink." Streaming computation of Delaunay triangulations," ACM Transactions on Graphics (TOG).New York, vol. 25,issue 23,pp. 1049-1056, July 2006.

[7] Sangyoon Lee,Chan-Ik Park,Chan-Mo Park." An Improved Parallel Algorithm for Delaunay Triangulation on Distributed Memory Parallel Computers," Proceedings of the 1997 Advances in Parallel and Distributed Computing Conference (APDC '97).Shanghai, pp.131-138, March 1997.